Medientechnik

Sommersemester 2016

Übung 02 (MVC, Observer & Co.)

Terminübersicht

Nr	Zeitraum	Thema
1	18.04. - 21.04.	Organisatorisches, Bildbearbeitung
2	09.05. - 12.05.	JavaFX Einführung (GUIs, Szenengraph)
3	**17.05. - 19.05.**	**Design Patterns: MVC, Observer**
4	23.05. - 25.05.	Bildfilter programmieren
5	30.05. - 02.06.	Videobearbeitung
6	06.06. - 09.06.	Video Streams mit JavaFX
7	20.06. - 23.06.	Audiobearbeitung
8	27.06. - 30.06.	Multimedia mit JavaFX

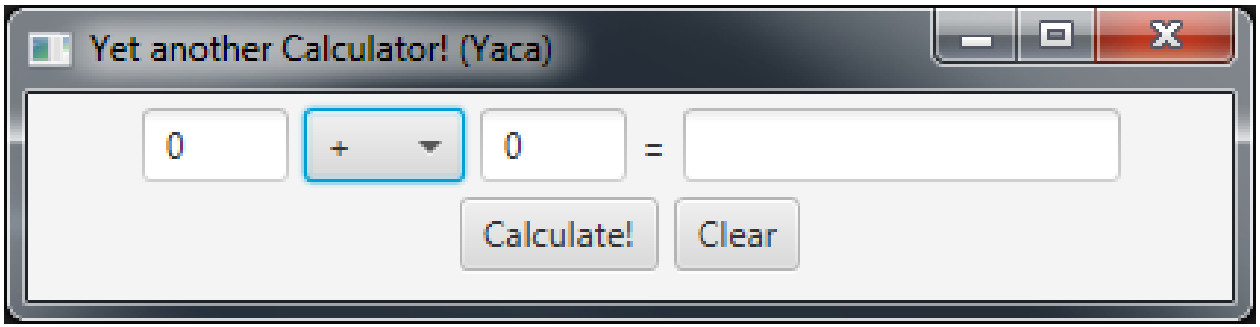
Agenda

- Wiederholung Übung 1
- Model-View Controller (MVC)
- EventHandler
- Observer Pattern

Yaca

- Letzte Übung:
  - GUI für sehr einfachen Taschenrechner
  - Unterschiedliche Elemente mit Hilfe von Panes und Boxes
  - Default Werte für einzelne Elemente setzen

- Source code jetzt online unter: http://www.medien.ifi.lmu.de/lehre/ss16/mt/uebung/ressourcen/mt_material02.zip



Warm-Up Quiz

- Was ist ein Szenengraph?
- Wofür braucht man “Container”?
- Was macht ein BorderPane?
- Was macht `primaryStage.setScene(scene);`?
- Welche Methode muss man aufrufen, damit die “Bühne” sichtbar wird?

Los geht’s!

1. Lieblings-Editor starten
2. Projekt der letzten Übung öffnen
3. Source Code evtl. übernehmen
   http://www.medien.ifi.lmu.de/lehre/ss16/mt/uebung/ressourcen/mt_material02.zip

Model View Controller (MVC)

- Software Design Pattern (Entwurfsmuster)
  - Wiederverwendbare Lösung für bekannte Programmierprobleme
  - Lediglich Beschreibung → Implementierung muss man selbst machen

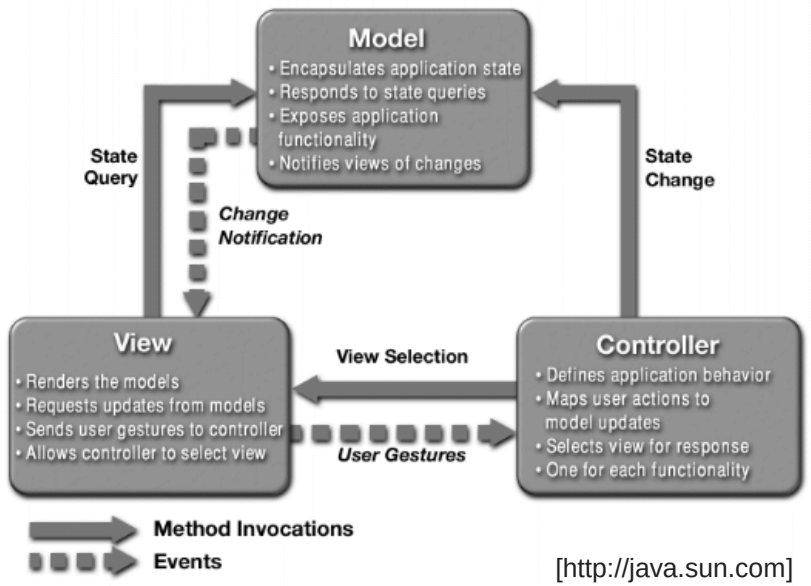
- “Separation of Concerns”
  - Flexibilität
  - Komponenten leicht austauschbar

- Weitere Varianten haben sich entwickelt (Model-View-Presenter, Model-View-ViewModel und viele andere)

https://en.wikipedia.org/wiki/Software_design_pattern

MVC Komponenten

- Model:
  - Speichert Daten
  - Gibt Auskunft über aktuelle Werte und notifiziert ggf.
- View:
  - “dumme Anzeige”
  - Wird vom Model benachrichtigt, wenn neue Daten anzuzeigen sind
  - Gibt Nutzereingaben erst mal an den Controller weiter
- Controller:
  - Legt das Verhalten des Programms fest.
  - Kann Änderungen von Werten im Model vornehmen.



[http://java.sun.com]

Model

Model.java

```
import javafx.stage.Stage;

public class Model {

    private Stage primaryStage = null;

    public Model (Stage primaryStage) {
        this.primaryStage = primaryStage;
    }

    public Stage getPrimaryStage() {
        return primaryStage;
    }
}
```

YacaView

YacaView.java

```
import …

public class YacaView {

    public YacaView() {

    }



    public void show(Stage stage){
        int padding = 5;
        double fieldWidth = 50;

        […]
    }
}
```

Controller

Controller.java

```
public class Controller {
    private YacaView yacaView;
    private Model model;

    public Controller(Model model) {
        this.model = model;
        this.yacaView = new YacaView();
    }

    public void showGUI(){
        this.yacaView.show(this.model.getPrimaryStage());
    }
}
```

Main

Main.java

```
import javafx.application.Application;
import javafx.stage.Stage;

public class Main extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    public void start(Stage primaryStage) throws Exception {
        Model model = new Model(primaryStage);
        Controller controller = new Controller(model);
        controller.showGUI();
    }
}
```

Daten und Funktionalität kapseln und zur Verfügung stellen

MODEL

Model – Instanzvariablen für Werte

```
import [...]

public class Model extends Observable {

    private Stage primaryStage = null;

    private double firstNumber,
            secondNumber,
            result;

    public Model(Stage primaryStage) {
        this.primaryStage = primaryStage;
        this.firstNumber = 0;
        this.secondNumber = 0;
        this.result = 0;
    }


    public Stage getPrimaryStage() {
        return primaryStage;
    }
}
```

Model – Methoden

```
public class Model {

    [...]

    public void calculateResult(){
        //TODO
    }

    public void reset() {
        this.firstNumber = 0;
        this.secondNumber = 0;
        this.result = 0;
    }

    [...]
}
```

Model – Operatoren

```
public class Model {

    private Stage primaryStage = null;

    private String currentOperator = null;
    private ObservableList<String> operators = null;

    private double firstNumber,
            secondNumber,
            result;

    public Model(Stage primaryStage) {
        this.primaryStage = primaryStage;
        this.firstNumber = 0;
        this.secondNumber = 0;
        this.result = 0;
        this.operators = FXCollections.observableArrayList(
                                "+", "-", "*", "/" );
        this.currentOperator = "+";
    }
}
```

Model – Getter/Setter

```
public class Model {

    [...]
    public ObservableList<String> getOperators() {
        return operators;
    }

    public double getFirstNumber() {
        return firstNumber;
    }

    public void setFirstNumber(double firstNumber) {
        this.firstNumber = firstNumber;
    }

    public double getSecondNumber() {
        return secondNumber;
    }

    public void setSecondNumber(double secondNumber) {
        this.secondNumber = secondNumber;
    }

    public void setCurrentOperator(String operator) {
        this.currentOperator = operator;
    }

    public Stage getPrimaryStage() {
        return primaryStage;
    }
}
```

Model – Ergebnis berechnen

```
public class Model {

    [...]

    public void calculateResult() {
        switch (this.currentOperator) {
            case "+":
                this.result = this.firstNumber + this.secondNumber;
                break;
            case "-":
                this.result = this.firstNumber - this.secondNumber;
                break;
            case "*":
                this.result = this.firstNumber * this.secondNumber;
                break;
            case "/":
                this.result = this.firstNumber / this.secondNumber;
                break;
        }
    }
    [...]
}
```

Anzeigen, Ereignisse weitergeben.

VIEW

YacaView – Instanziierung

YacaView.java

```
public class YacaView {

    private TextField resultField;
    private TextField firstNumberField;
    private TextField secondNumberField;

    private Button calculateButton;
    private Button resetButton;

    private ComboBox<String> operatorBox;

    public YacaView() {
        this.firstNumberField = new TextField();
        this.secondNumberField = new TextField();

        this.resultField = new TextField();


        this.calculateButton = new Button("Calculate!");
        this.resetButton = new Button("Clear");

        this.operatorBox = new ComboBox<String>();
    }

    [...]
}
```

YacaView – Getter/Setter

```
public class YacaView {

    [...]

    public double getFirstValue() {
        return Double.parseDouble(this.firstNumberField.getText());
    }
    public double getSecondValue() {
        return Double.parseDouble(this.secondNumberField.getText());
    }

    public Button getCalculateButton() {
        return this.calculateButton;
    }

    public Button getResetButton() { return this.resetButton;  }

    public ComboBox getOperatorBox() {return this.operatorBox; }

    // Setter!
    public void setOperators(ObservableList<String> operators) {
        this.operatorBox.setItems(operators);
    }
}
```

Zwischenüberlegungen

- Zu diesem Zeitpunkt sind Model, View und Controller noch voneinander losgelöst

- Jetzt fehlen also noch diese Punkte:
  - View soll Nutzereingaben an den Controller weitergeben
  - Controller soll auf solche Ereignisse reagieren
  - Model soll die Berechnung ausführen
  - View soll das Ergebnis anzeigen

YacaView – Event Handling 1

```
public class YacaView {
    […]

    private EventHandler<ActionEvent> handler;

    public YacaView(EventHandler<ActionEvent> handler) {

        this.handler = handler;

        this.firstNumberField = new TextField();
        this.secondNumberField = new TextField();

        this.resultField = new TextField();


        this.calculateButton = new Button("Calculate!");
        this.resetButton = new Button("Clear");

        this.operatorBox = new ComboBox<String>();
    }
}
```

YacaView – Event Handling 2

```
public class YacaView {

    […]


    public void show(Stage stage){

        […]
        this.calculateButton.setOnAction(this.handler);
        this.resetButton.setOnAction(this.handler);
        this.operatorBox.setOnAction(this.handler);

        […]
    }

}
```

Ablaufsteuerung

CONTROLLER

Controller - Überlegungen

- Controller muss auf Ereignisse reagieren:
  → Implementiert die Schnittstelle EventHandler<ActionEvent>

- Muss erkennen können, welches Event eingetreten ist
  → “Quelle” des Ereignisses liefert Auskunft darüber

- Quelle der Ereignisse wird mit Instanzvariablen (Buttons, ComboBox) der YacaView verglichen und entsprechend reagiert.

Controller - EventHandler Schnittstelle

```
import javafx.event.ActionEvent;
import javafx.event.EventHandler;

public class Controller implements EventHandler<ActionEvent> {
    private YacaView yacaView;
    private Model model;

    public Controller(Model model) {
        this.model = model;
        this.yacaView = new YacaView(this);
    }

    @Override
    public void handle(ActionEvent event) {
        // TODO
    }

    public void showGUI(){
        this.yacaView.setOperators(this.model.getOperators());
        this.yacaView.show(this.model.getPrimaryStage());
    }

}
```

Controller - handle Methode

```
public class Controller implements EventHandler<ActionEvent> {
    […]
    public void handle(ActionEvent event) {
        Object source = event.getSource();
        ComboBox operatorBox;
        if (source.equals(this.yacaView.getOperatorBox())) {
            operatorBox = (ComboBox) source;
            this.model.setCurrentOperator((String) operatorBox
                    .getSelectionModel()
                    .getSelectedItem());
        }
        else if(source.equals(this.yacaView.getCalculateButton())){
            this.model.setFirstNumber(this.yacaView.getFirstValue());
            this.model.setSecondNumber(this.yacaView.getSecondValue());
            this.model.calculateResult();
        }
        else if(source.equals(this.yacaView.getResetButton())){
            this.model.reset();
        }
    }
}
```

Ergebnis anzeigen?

- Aktuell: Das Model berechnet jetzt wie gewollt das Ergebnis.
- Nächster Schritt: Ergebnis anzeigen. Aber wie?
- 1. Ansatz: in der .handle() Methode noch auf das resultField von der YacaView zugreifen und den entsprechenden Wert setzen.
- 2. Ansatz: die YacaView “beobachtet” das Ergebnis und reagiert dann immer wenn es sich ändert

Observer Pattern!

Oh! Noch ein Design Pattern!

OBSERVER PATTERN

Wichtigste Komponenten

- `java.util.Observable`
  - Klasse, von der man Hilfsmethoden erben kann
  - `.setChanged()` deutet an, dass sich das Observable in einem neuen Zustand befinden, der relevant für alle Observer sein könnte
  - `.notifyObservers()` benachrichtigt alle, die sich für dieses Observable registriert haben.
- `java.util.Observer`
  - Schnittstelle, die das Beobachten eines Observable ermöglicht
  - `.update()` Methode muss implementiert werden. Hier kann man auf Änderungen des Observables reagieren.

Model anpassen

```
import java.util.Observable;

public class Model extends Observable{

    […]
    public void calculateResult(){
        switch (this.currentOperator) {
            […]
        }
        this.setChanged();
        this.notifyObservers(this.result);
    }

    public void reset() {
        this.firstNumber = 0;
        this.secondNumber = 0;
        this.result = 0;
        this.setChanged();
        this.notifyObservers("reset");
    }
}
```

View anpassen

```
import java.util.Observable;
import java.util.Observer;

public class YacaView implements Observer{
    […]
    public void update(Observable o, Object arg) {
        if(arg.equals("reset")) {
            this.secondNumberField.setText(String.valueOf(0));
            this.firstNumberField.setText(String.valueOf(0));
            this.resultField.setText("");
        }
        else {
            this.resultField.setText(arg.toString());
        }
    }
    […]
}
```

Controller anpassen

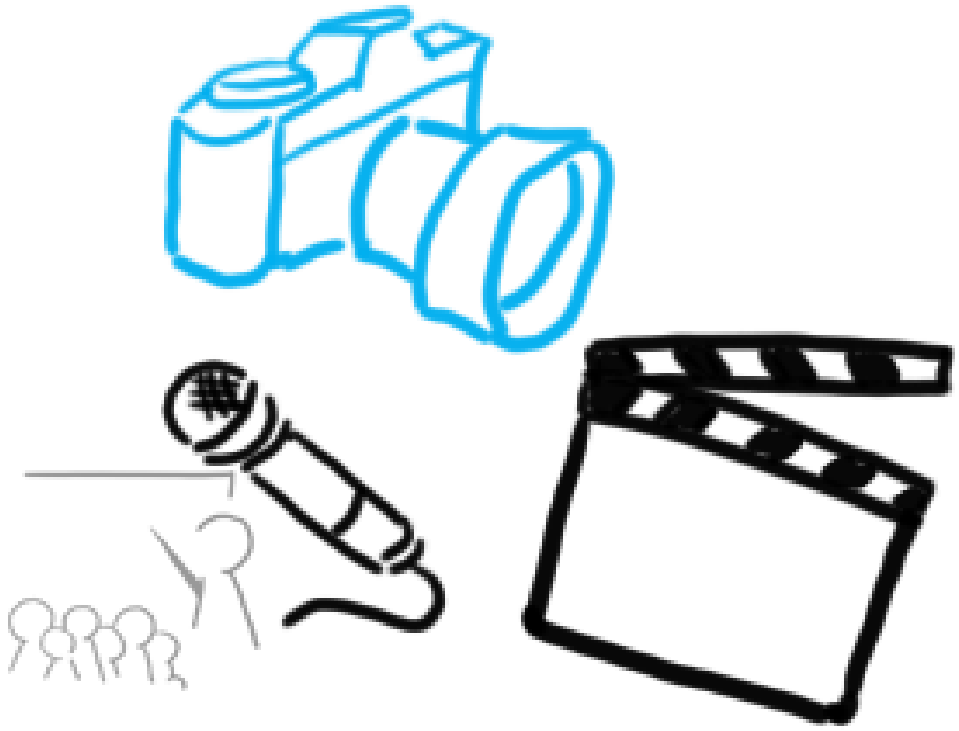
```
public class Controller implements EventHandler<ActionEvent> {

    […]

    public void showGUI(){
        this.yacaView.setOperators(this.model.getOperators());
        this.yacaView.show(this.model.getPrimaryStage());
        this.model.addObserver(this.yacaView);
    }
}
```

Wrap-Up Quiz

1. In welcher Klasse wird der Observer beim Modell registriert?
2. Welche Methode der Schnittstelle EventHandler muss man implementieren?
3. Wie wird ein EventHandler einem Element hinzugefügt?
4. Ist Observable eine Klasse oder eine Schnittstelle?
5. Welche beiden Methoden müssen aufgerufen werden, wenn sich etwas am Modell geändert hat?
6. Wann kann es zu einer NumberFormatException kommen?
7. Was bewirkt die folgende Zeile?
   Model m = (Model) someObject;
8. Warum wurden in unserem Beispiel die TextFelder der YacaView zu Instanzvariablen gemacht?



Vielen Dank!

WELCHE FRAGEN HABT IHR?